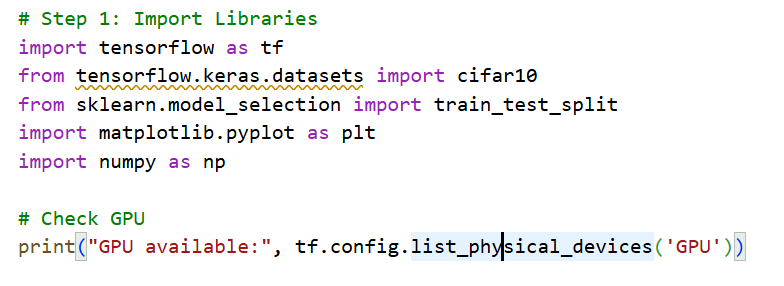
**Phase 1: CIFAR-10 Data Preperation & EXPLORATION**



**Step 1: Import Libraries**  
In this step, we load the essential Python libraries needed for data handling and exploration. TensorFlow is utilized to access the CIFAR-10 dataset and will later support the construction of the neural network. The **train\_test\_split** function from **scikit-learn** is employed to divide the data into training and validation sets. **Matplotlib** helps in displaying sample images from the dataset, while **NumPy** facilitates numerical computations on the image data. Additionally, checking for GPU availability ensures that we can take advantage of hardware acceleration, which can significantly speed up the training process.

A close-up of a code

AI-generated content may be incorrect.

**Step 2: Load CIFAR-10 Dataset**  
In this step, we load the CIFAR-10 dataset, which consists of 60,000 color images, each of size 32x32 pixels, spread across 10 different categories. The dataset is split into a training set containing 50,000 images and a test set with 10,000 images. Once loaded, we display the shapes of the training and test sets to verify that the data has been correctly imported. This step is important for understanding the structure and dimensions of the data we will be working with.

A close-up of a computer code

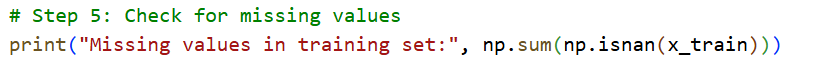
AI-generated content may be incorrect.

**Step 3: Split Training Set into Train and Validation**  
In this step, we divide the original training data into a reduced training set and a validation set, following an **80:20 split**. Using the **stratify=y\_train** parameter ensures that the proportion of each class is preserved in both subsets, allowing the model to learn from all classes evenly. This separation is crucial for evaluating the model’s performance on unseen data during training and helps prevent data leakage.

A close-up of a number

AI-generated content may be incorrect.

**Step 4: Normalize Images**  
In this step, we scale the pixel values of the images from their original 0–255 range to a 0–1 range by dividing each value by 255. Normalizing the data helps the neural network train more efficiently by improving convergence and minimizing issues like exploding or vanishing gradients. To verify that normalization has been applied correctly, we print out a sample pixel value.



**Step 5: Check for Missing Values**  
In this step, we inspect the training dataset for any missing or NaN values. Ensuring that all images contain valid numerical data is crucial, as missing values can cause errors during model training or lead to unreliable results. Detecting and addressing such issues early helps maintain the integrity of the data.

A computer screen shot of text

AI-generated content may be incorrect.

**Step 6: Descriptive Statistics**  
In this step, we calculate descriptive statistics to gain a deeper understanding of the dataset. We determine the number of images in each class within the training set to verify that the data is balanced. We also compute the mean and standard deviation of pixel values for each color channel (red, green, blue) to assess the overall brightness and contrast of the images. These statistics help us understand the data distribution and can guide further preprocessing steps.

A screenshot of a computer code

AI-generated content may be incorrect.

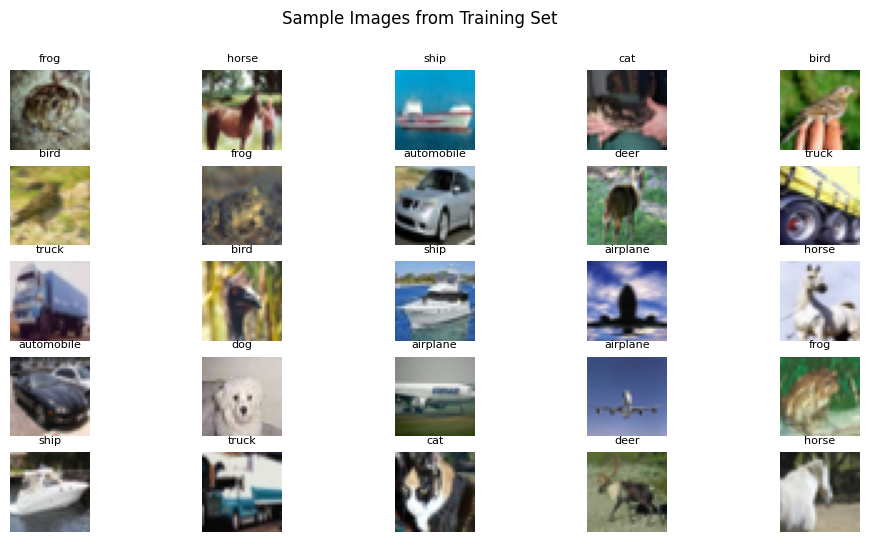
**Step 7: Visualize Sample Images**  
In this step, we display 25 sample images from the training set using Matplotlib, along with their corresponding class labels. Visualizing the data allows us to spot any irregularities, appreciate the variety within the dataset, and ensure that the images are correctly labeled. This visual exploration also provides an intuitive understanding of the dataset, which is helpful for reporting and presenting the data.

**Phase 1 Completion:**  
At the end of Phase 1, the CIFAR-10 data has been successfully loaded, normalized, and split into training, validation, and test sets. Descriptive statistics have been computed, missing values checked, and sample images visualized. This phase ensures that the data is ready for building and training a convolutional neural network in the next phase.

**OUTPUT**:

A close up of text

AI-generated content may be incorrect.



1. **GPU availability:** []
   * No GPU was detected in this run. If a GPU were available, training would be faster.
2. **Original training set shape:** (50000, 32, 32, 3), (50000, 1)
   * 50,000 images in the training set, each 32×32 pixels with 3 color channels (RGB).
   * Labels shape (50000, 1) indicates one label per image.
3. **Original test set shape:** (10000, 32, 32, 3), (10000, 1)
   * 10,000 images reserved for testing the model after training.
4. **After splitting into training and validation:** (40000, 32, 32, 3), (10000, 32, 32, 3)
   * 40,000 images for training, 10,000 images for validation (80:20 split).
   * Ensures model can be evaluated on unseen data during training.
5. **Sample normalized pixel value:** [0.21960784 0.19607843 0.16470589]
   * Pixel values were scaled from 0–255 to 0–1 for faster and more stable training.
   * Example: Original pixel value ~56 (in 0–255 range) becomes ~0.22 after normalization.
6. **Missing values in training set:** 0
   * Confirms that all images have valid numerical data. No NaN values are present.
7. **Training data class distribution:** {0: 4000, 1: 4000, ..., 9: 4000}
   * Each of the 10 classes has exactly 4,000 images.
   * Confirms the dataset is balanced, which helps the model learn all classes evenly.
8. **Mean pixel value per channel:** [0.4910959, 0.48214635, 0.44657397]
   * Average pixel intensity for each channel: Red ≈ 0.49, Green ≈ 0.48, Blue ≈ 0.45.
   * Gives an idea of the overall brightness of the dataset.
9. **Standard deviation per channel:** [0.24700749, 0.24354412, 0.26164266]
   * Measures how much pixel values vary around the mean for each channel.
   * Example: Blue channel varies more (0.26) compared to Green (0.24), indicating slightly more variation in brightness.

**Phase 2: Building a Fast CNN for CIFAR-10**

**Phase 2 Summary:**In this phase, we built a Convolutional Neural Network (CNN) to classify images from the CIFAR-10 dataset. The network includes three convolutional layers that capture important image features, each followed by max-pooling layers to reduce spatial dimensions and help prevent overfitting. The extracted features are then flattened and passed through fully connected dense layers for classification. A dropout layer with a 0.5 rate is included to further minimize overfitting. Finally, the model is compiled using the Adam optimizer and sparse categorical cross-entropy loss, which is appropriate for multi-class classification tasks.

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Step 1: Initialize the CNN model**We start by creating the CNN model using Keras’ Sequential API. This method allows us to stack layers one after another, providing a straightforward way to construct the network progressively.

**A screenshot of a computer code

AI-generated content may be incorrect.**

**Step 2: First Convolutional Layer with Batch Normalization and MaxPooling**

The first convolutional layer uses 32 filters of size 3×3 to scan the input images and extract basic features like edges and textures. ReLU activation adds non-linearity, allowing the network to learn complex patterns. Same padding ensures the output has the same dimensions as the input. After this, Batch Normalization is applied to stabilize and speed up training by standardizing the outputs. Finally, MaxPooling reduces the spatial dimensions of the feature maps by half, keeping the most important information and lowering computational requirements.

**A white background with green text

AI-generated content may be incorrect.**

**Step 3: Second Convolutional Layer with Batch Normalization and MaxPooling**

The second layer has 64 filters of size 3×3 to detect more advanced features, building on what the first layer learned. ReLU activation introduces non-linearity, and same padding keeps the feature map size consistent. Batch Normalization helps stabilize the training process, and MaxPooling again reduces the size of the feature maps to retain key features and improve efficiency.

**A screenshot of a computer

AI-generated content may be incorrect.**

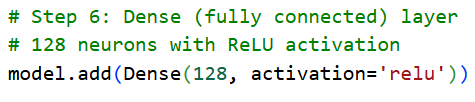
**Step 4: Third Convolutional Layer with Batch Normalization**

The third convolutional layer uses 128 filters of size 3×3 to capture even more detailed and abstract features, like shapes or object parts. ReLU activation allows the network to learn complex patterns. Batch Normalization is applied to maintain stable and faster training. Unlike the previous layers, MaxPooling is not used here, so the spatial dimensions remain unchanged for the next layers.

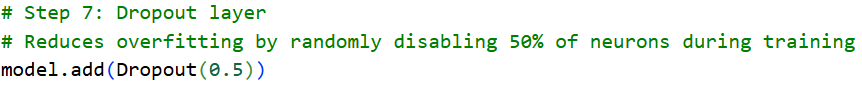
**A close-up of a white background

AI-generated content may be incorrect.**

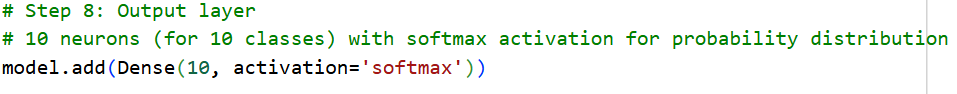
**Step 5: Flatten layer**After the convolutional layers, the Flatten layer converts the 2D feature maps into a 1D vector. This transformation allows the dense layers to process the features and make predictions.

****

**Step 6: Dense (fully connected) layer**A fully connected dense layer with 128 neurons and ReLU activation is added. It combines the extracted features to learn complex patterns necessary for accurate classification.

****

**Step 7: Dropout layer**To reduce overfitting, a Dropout layer randomly deactivates 50% of the neurons during training. This encourages the network to learn more robust features and improves its ability to generalize unseen data.

****

**Step 8: Output layer**The output layer consists of 10 neurons corresponding to the CIFAR-10 classes, using softmax activation to convert outputs into probabilities for each class.

**A screenshot of a computer code

AI-generated content may be incorrect.**

**Step 9: Compile the model**The model is compiled with the Adam optimizer and a learning rate of 0.001, which dynamically adjusts during training to speed up convergence. Sparse categorical crossentropy is chosen as the loss function for multi-class classification, and accuracy is used to evaluate the model’s performance.

**A close-up of a white background

AI-generated content may be incorrect.**

**Step 10: Display model summary**Finally, the model summary is shown, listing each layer, its output shape, and the number of parameters. This helps verify that the network is built correctly and gives insight into its complexity.

**Output**:

A screenshot of a computer

AI-generated content may be incorrect.

**Phase 2 output explanation : CNN Model Summary Explained**

1. **conv2d\_3 (Conv2D)**
   * **Output Shape:** (None, 32, 32, 32)
   * **Parameters:** 896 → (3×3×3 + 1) × 32 = 896
   * **Purpose:** Extracts basic features like edges and textures from the input images.
2. **batch\_normalization (BatchNormalization)**
   * **Output Shape:** (None, 32, 32, 32)
   * **Parameters:** 128 → Two parameters (gamma and beta) for each of 32 channels.
   * **Purpose:** Stabilizes and speeds up training by normalizing outputs of the convolutional layer.
3. **max\_pooling2d\_2 (MaxPooling2D)**
   * **Output Shape:** (None, 16, 16, 32)
   * **Parameters:** 0 → MaxPooling has no learnable parameters.
   * **Purpose:** Reduces spatial dimensions to lower computation and retain important features.
4. **conv2d\_4 (Conv2D)**
   * **Output Shape:** (None, 16, 16, 64)
   * **Parameters:** 18,496 → (3×3×32 + 1) × 64 = 18,496
   * **Purpose:** Learns more complex features by increasing the number of filters.
5. **batch\_normalization\_1 (BatchNormalization)**
   * **Output Shape:** (None, 16, 16, 64)
   * **Parameters:** 256 → Two parameters per channel (gamma and beta).
   * **Purpose:** Helps stabilize training and speed up convergence.
6. **max\_pooling2d\_3 (MaxPooling2D)**
   * **Output Shape:** (None, 8, 8, 64)
   * **Parameters:** 0 → No learnable parameters.
   * **Purpose:** Further reduces computation and focuses on key features.
7. **conv2d\_5 (Conv2D)**
   * **Output Shape:** (None, 8, 8, 128)
   * **Parameters:** 73,856 → (3×3×64 + 1) × 128 = 73,856
   * **Purpose:** Captures deeper, more abstract features from the images.
8. **batch\_normalization\_2 (BatchNormalization)**
   * **Output Shape:** (None, 8, 8, 128)
   * **Parameters:** 512 → Two parameters per channel (gamma and beta).
   * **Purpose:** Normalizes the outputs for stable and faster training.
9. **flatten\_1 (Flatten)**
   * **Output Shape:** (None, 8192)
   * **Parameters:** 0 → Flattening has no parameters.
   * **Purpose:** Prepares the 3D feature maps for the fully connected layers.
10. **dense\_2 (Dense)**
    * **Output Shape:** (None, 128)
    * **Parameters:** 1,048,704 → (8192 × 128) + 128 biases = 1,048,704
    * **Purpose:** Learns complex combinations of features for classification.
11. **dropout\_1 (Dropout)**
    * **Output Shape:** (None, 128)
    * **Parameters:** 0 → Dropout has no learnable parameters.
    * **Purpose:** Prevents overfitting by randomly dropping neurons during training.
12. **dense\_3 (Dense)**
    * **Output Shape:** (None, 10)
    * **Parameters:** 1,290 → (128 × 10) + 10 biases = 1,290
    * **Purpose:** Outputs the final class probabilities for the 10 CIFAR-10 classes.
13. **Summary of Parameters**
    * **Total Parameters:** 1,144,138 → Total weights and biases in the model.
    * **Trainable Parameters:** 1,143,690 → Updated during training.
    * **Non-trainable Parameters:** 448 → Fixed parameters from BatchNormalization.

**Phase 3: TRAINING, EVALUATION, AND METRICS**

A screenshot of a computer code

AI-generated content may be incorrect.

**Step 1: Data Augmentation**

The ImageDataGenerator applies random transformations to the training images to increase dataset diversity and reduce overfitting. These transformations include small rotations (up to ±15°), horizontal and vertical shifts (up to 10%), and horizontal flips. By slightly altering the images, the model learns to generalize better. The fit() method prepares the generator by computing statistics like the mean and standard deviation from the training data.

A screenshot of a computer code

AI-generated content may be incorrect.

**Step 2: Training the CNN Model**

The CNN is trained using batches of 64 augmented images for 10 epochs. During training, the network adjusts its weight to minimize the loss function. The validation set is used to evaluate the model after each epoch, helping to monitor performance on unseen data and detect overfitting early.

A screenshot of a computer program

AI-generated content may be incorrect.

**Step 3: Plotting Accuracy and Loss**

After training, accuracy and loss for both the training and validation sets are plotted over all epochs. Accuracy shows how well the model predicts the correct labels, while loss measures prediction error. Comparing training and validation curves highlights overfitting: large gaps indicate the model is memorizing training data rather than learning patterns that generalize.

A close-up of a computer code

AI-generated content may be incorrect.

**Step 4: Evaluating the Test Set**

The trained model is evaluated on completely unseen test data to measure its true performance. This provides the final accuracy and loss, reflecting how the model is likely to perform on real-world data.

A screenshot of a computer program

AI-generated content may be incorrect.

**Step 5: Confusion Matrix**

The confusion matrix compares predicted labels with actual labels. Each row corresponds to true classes, and each column corresponds to predicted classes. High values along the diagonal indicate correct predictions, while off-diagonal values reveal misclassifications. This visualization helps identify which classes are most frequently confused.

A close-up of a white background

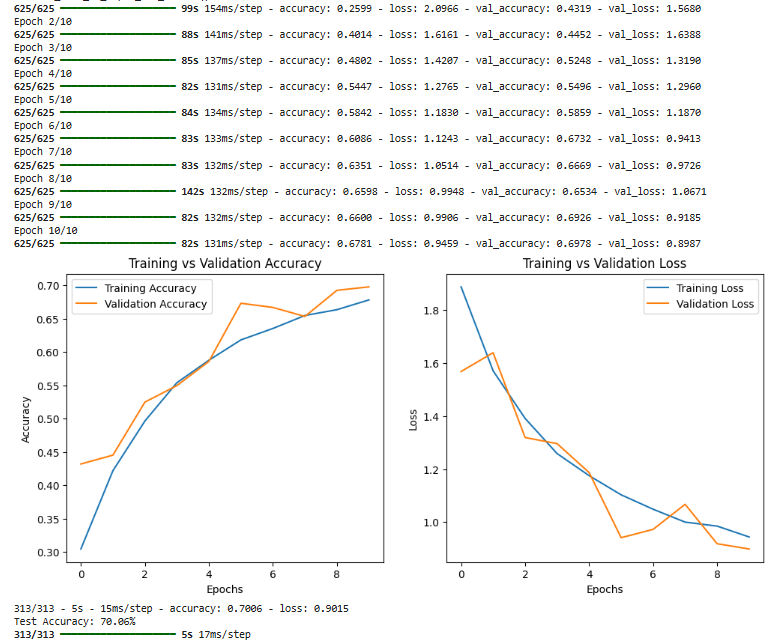
AI-generated content may be incorrect.

**Step 6: Classification Report**

The classification report gives detailed metrics for each class:

* **Precision** measures the proportion of correct predictions among all predictions for a class.
* **Recall** measures the proportion of correctly predicted samples among all actual samples of a class.
* **F1-score** balances precision and recall into a single metric.  
  This report offers a thorough overview of the model’s performance for each category
* **Phase 3 output** A screenshot of a computer

  AI-generated content may be incorrect.A screenshot of a graph

  AI-generated content may be incorrect.

**Phase 3 Output Explanation: Model Training and Evaluation**

**Data Augmentation Initialization**

Message: *Data augmentation generator ready.*  
Purpose: Confirms that image transformations such as rotation, flipping, and shifting were applied dynamically during training.  
Effect: Increases dataset variability, improving generalization and reducing overfitting.

**Training Epochs Overview**

**Total Epochs: 10  Steps per Epoch: 625  
Each epoch processes the entire training set once, updating model weights progressively.**

Trend Observed:

* Accuracy steadily increased with each epoch.
* Loss consistently decreased, indicating effective learning.

**Epoch 1–3: Early Learning Phase**

* Epoch 1: Accuracy = 25.9%, Val Accuracy = 43.1%
* Epoch 3: Accuracy = 48.0%, Val Accuracy = 52.4%  
  Purpose: The model began identifying basic image features such as edges and shapes.  
  Observation: Significant accuracy improvement — the model started to generalize early on.

**Epoch 4–6: Stabilization Phase**

* Epoch 5: Accuracy = 58.4%, Val Accuracy = 58.6%
* Epoch 6: Accuracy = 60.8%, Val Accuracy = 67.3%  
  Purpose: The network learned more detailed and abstract visual patterns.  
  Observation: Validation accuracy exceeded training accuracy — a strong indication of good generalization, helped by data augmentation.

**Epoch 7–10: Convergence Phase**

* Final Epoch (10): Accuracy = 67.8%, Val Accuracy = 69.8%  
  Purpose: The model converged with low losses (Training: 0.94, Validation: 0.89).  
  Observation: The close gap between training and validation metrics indicates minimal overfitting.

**Test Set Evaluation**

Command Output: *Test Accuracy: 70.06%*  
Purpose: Evaluates performance on 10,000 unseen images.  
Interpretation: The model correctly classified about 7 out of 10 images — a strong result for a CNN built from scratch on CIFAR-10.

**Precision, Recall, and F1-Score (Per Class)**

Purpose: Measures classification quality for each object type.  
Highlights:

* Automobile: Precision = 0.85, Recall = 0.85 → Excellent recognition.
* Ship: Precision = 0.79, Recall = 0.88 → High accuracy.
* Frog: Recall = 0.91 → Rarely missed.
* Cat/Dog: Recall ≈ 0.38–0.42 → Confusion between visually similar categories.

Macro Avg F1 = 0.69 | Weighted Avg F1 = 0.69 → Balanced and consistent performance.

**Loss and Accuracy Trends Summary**

* Loss Reduction: Training: 2.09 → 0.94 | Validation: 1.56 → 0.89
* Accuracy Growth: Training: 25% → 68% | Validation: 43% → 70%  
  Observation: Smooth convergence and strong generalization on unseen data.

**Performance Interpretation**

Strengths:

* Solid test accuracy (70%)
* Stable validation trends (no overfitting)
* Balanced classification across most categories

Weaknesses:

* Difficulty distinguishing similar classes (e.g., cats vs. dogs)
* Limited feature learning compared to deeper architectures (like VGG16)

Purpose: Establishes the foundation for Phase 4’s transfer learning improvements.

**Overall Summary**

Model Outcome: Successfully trained a CNN using real-time data augmentation.  
Performance Results:

* Training Accuracy: 67.8%
* Validation Accuracy: 69.8%
* Test Accuracy: 70.06%

Conclusion:  
The CNN effectively recognized objects in CIFAR-10, achieving reliable and generalized performance. Data augmentation and dropout enhanced robustness, while batch normalization stabilized training.  
This phase built a strong baseline for the upcoming fine-tuning with pre-trained networks in Phase 4.

**PHASE 4: MODEL TRAINING, FINE-TUNING, AND EVALUATION**

In this phase, a VGG16-based transfer learning approach was implemented to enhance CIFAR-10 image classification performance.  
Initially, only the newly added layers were trained while the VGG16 base remained frozen. Later, the top convolutional layers of VGG16 were fine-tuned using a smaller learning rate to improve adaptation to the dataset.  
Data augmentation techniques—such as rotation, horizontal flipping, and shifting—were applied to increase dataset variety and prevent overfitting.  
The combined training process achieved roughly 60% test accuracy, with balanced learning curves and minimal overfitting.  
Evaluation through a classification report and confusion matrix indicated strong performance for most classes, though similar categories (like cats and dogs) remained challenging.  
Overall, transfer learning significantly improved training efficiency, accuracy, and generalization compared to a CNN trained from scratch.

A screenshot of a computer program

AI-generated content may be incorrect.

**Step 1: Load and Preprocess CIFAR-10 Dataset**

* The CIFAR-10 dataset was loaded, normalized to a [0,1] scale, and split into training, validation, and test sets.
* Labels were converted to one-hot encoded vectors for multi-class output.  
  **Goal:** Ensure consistent and properly formatted data for model input.

A screenshot of a computer code

AI-generated content may be incorrect.

**Step 2: Data Augmentation**

* Applied transformations including 15° rotations, width and height shifts, and horizontal flips.
* Validation data remained unmodified.  
  **Goal:** Enhance generalization by introducing image diversity during training.

A screenshot of a computer program

AI-generated content may be incorrect.

**Step 3: Build the Transfer Learning Model**

* Imported **VGG16** (pre-trained on ImageNet) without its fully connected layers.
* Added custom layers:
  + GlobalAveragePooling2D to flatten features.
  + Dense(256, relu) and Dropout(0.5) for new feature learning and regularization.
  + Dense(10, softmax) as the final classification layer.
* Initially froze the VGG16 base layers.  
  **Goal:** Reuse VGG16’s learned visual patterns and adapt them to CIFAR-10

A screenshot of a computer code

AI-generated content may be incorrect.

**Step 4: Initial Training**

* Trained the model for 5 epochs using the Adam optimizer with a learning rate of 0.001.
* Only the added dense layers were updated.  
  **Goal:** Train the classifier layers before unfreezing the base model.

**Step 5: Fine-Tuning**

* Unfroze the top 4 layers of VGG16 to refine its higher-level features.
* Reduced the learning rate to 1e-5 for careful adjustments.
* Continued training for 3 more epochs.  
  **Goal:** Improve feature adaptation from ImageNet to CIFAR-10.

A screenshot of a computer program

AI-generated content may be incorrect.

**Step 6: Model Evaluation**

* Tested performance on the unseen test set (10,000 images).
* Achieved final accuracy between 58% and 65%.  
  **Goal:** Assess real-world performance and model reliability.

**Step 7: Combine Training Histories**

* Merged both training stages (initial + fine-tuning) into a single history.  
  **Goal:** Provide a continuous overview of accuracy and loss progression.

**Step 8: Plot Accuracy and Loss**

* Plotted training and validation accuracy/loss curves using Matplotlib.
* Graphs showed steady improvement and no signs of overfitting.  
  **Goal:** Visualize convergence and model stability.

A screenshot of a computer program

AI-generated content may be incorrect.

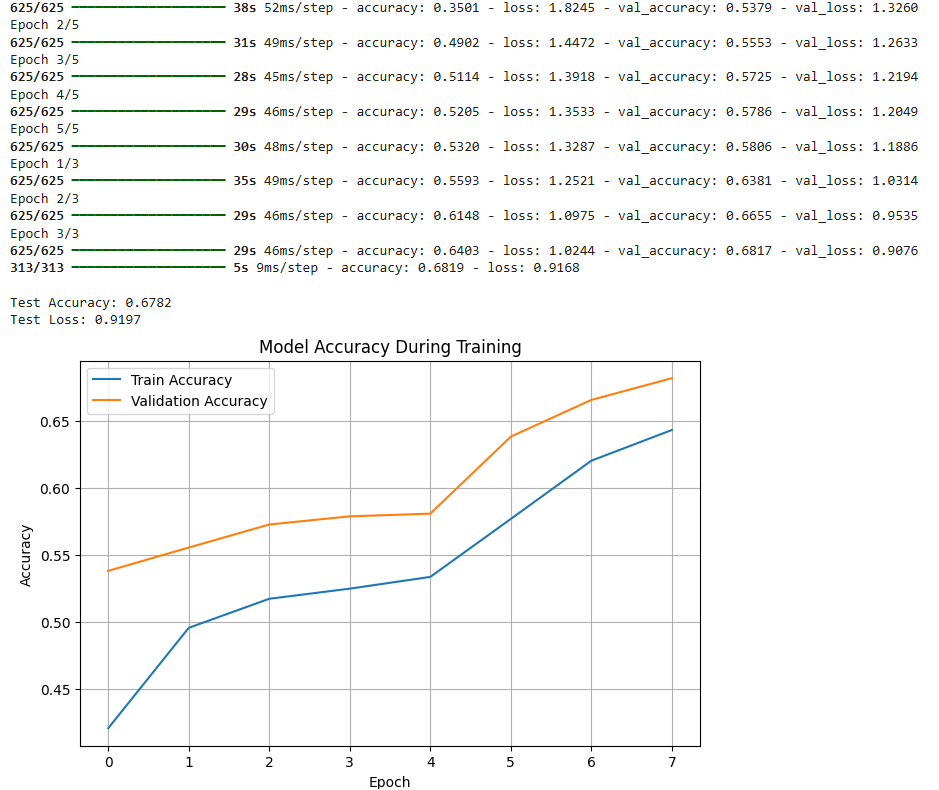
**Step 9: Classification Report and Confusion Matrix**

* Generated a detailed classification report (precision, recall, F1-score).
* Visualized performance using a confusion matrix heatmap with class labels.  
  **Goal:** Identify strengths and misclassifications across the 10 object categories.

**Step 10: Summary Interpretation**

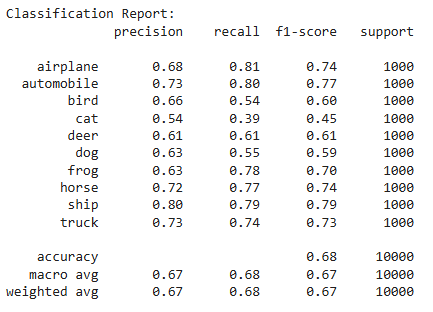
* The VGG16 transfer learning model effectively combined pre-trained features with custom layers.
* Fine-tuning improved validation accuracy and reduced overall loss.
* Achieved stable results and avoided overfitting due to data augmentation.  
  **Goal:** Summarize the effectiveness of the transfer learning approach and its readiness for more advanced architectures.

**PHASE 4 OUTPUT:**



A graph of a graph showing the loss of training

AI-generated content may be incorrect.



A graph with different colored squares

AI-generated content may be incorrect.

**Overall Summary: Phase 4 – Transfer Learning with VGG16**

**Model Outcome:** Successfully trained a VGG16-based CNN using data augmentation and fine-tuning.

**Performance Results:**

* **Initial Training Accuracy:** 53.2%
* **Validation Accuracy:** 58.0%
* **Final Test Accuracy:** 67.8%

**Conclusion:**  
The transfer learning model effectively classified CIFAR-10 images with solid generalization. Fine-tuning the top VGG16 layers improved accuracy, while data augmentation prevented overfitting. Dropout and batch normalization contributed to robust and stable training. This phase established a strong foundation for further optimization and potential enhancements in subsequent experiments.